VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Metody detekce pohybu ve vidosekvencích

Detection of movement in Video Sequences

2010 Martin Klapetek

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 6. května 2010 .....

Rád bych na tomto místě poděkoval Ing. Janu Gaurovi za odborné vedení jak při implementaci, tak při psaní samotného textu práce. Dále Bc. Danielu Sochorovi za pomoc při obstarávání materiálů popisujících používané metody v oblasti modelování pozadí. Také bych rád vyjádřil díky všem lidem, kteří tráví svůj volný čas připojení ke kanálu #opencl na serveru irc.feenode.net a ochotně pomáhají všem začínajícím s technologii OpenCL.

Abstrakt

Tato práce popisuje základy modelování pozadí videosekvencí a následnou segmentaci popředí. Je zde krátce rozebráno několik nejčastěji používaných metod v této oblasti počítačového vidění. Dále je detailně popsána základní rekurzivní multimodální metoda mixtura gausiánů. Jsou také zmíněny její běžné problémy při reálném nasazení a jejich možné řešení. Součástí této práce je implementace této metody v jazyce C s použitím knihovny OpenCV a také implementace v otevřeném výpočetním prostředí OpenCL. Způsob implementace obou variant algoritmu je podrobně popsán včetně nutných změn pro implementaci v OpenCL. U obou variant byl proveden test výkonu na dvou videosekvencích a výsledné hodnoty krátce okomentovány. V závěru práce je pak algoritmus mixtury gausiánů zhodnocen a jsou navrženy dodatečné úpravy pro implementaci.

Klíčová slova: segmentace popředí, modelování pozadí, mixtura gausiánů, OpenCV, OpenCL

Abstract

This thesis describes basics of background modeling of video sequences and its further foreground segmentation. Several mostly used methods in this part of computer vision are shortly described. Next part describes in detail basic recursive multimodal version of mixture of gaussians method. Mentioned are its common problems when used in real life and their possible solution. Part of this work is implementation of this method in C language with usage of OpenCV library and also an implementation in open computing environment OpenCL. Process of both implementations of this algorithm is described in detail, including all necessary changes for implementation in OpenCL. Both variants were tested for performance on two video sequences and the resulsts are shortly discussed. The final part then evaluates the mixture of gaussians method and proposes some additions for the implementation.

Keywords: foreground segmentation, background modeling, mixture of gaussians, OpenCV, OpenCL

Obsah

1 Úvod **5**

2 Oddělení popředí od pozadí **7**
2.1 Preprocessing 7
2.2 Modelování pozadí 8
2.3 Detekce popředí 12
2.4 Validace dat 12

3 Mixtura Gausiánů **14**
3.1 Teorie 14
3.2 Dodatečné rozšíření algoritmu 18

4 Implementace **20**
4.1 Použité technologie 20
4.2 Implementace 22

5 Testování **27**
5.1 Testování 1. videosekvence 28
5.2 Testování 2. videosekvence 29

6 Závěr **30**

7 Literatura **31**

Seznam tabulek

1 Počáteční parametry algoritmu 23

Seznam obrázků

1 Způsob, jakým vidí počítač 6
2 Proces segmentace pozadí 8
3 Příklad rozložení pravděpodobnosti pixelu, rovnice (2). $K = 3, I_t \in \{0, 1, ..., 255\}, \omega_k = \{0, 2, 0, 2, 0, 6\}, \mu_k = \{80, 100, 200\} a \sigma_k = \{20, 5, 10\}$. 15
4 A posteriori pravděpodobnosti, rovnice (6), vykresleny jako funkce I_t pro každé $k = 1, 2, 3$. Použity jsou stejné parametry jako v obr. 3 17
5 Příklad vzniklých děr v masce popředí, vlevo zpracovávaný snímek videosekvence, vpravo maska popředí vytvořená algoritmem z kapitoly 4 . . . 19
6 Workflow diagram implementovaného algoritmu Mixtury Gausiánů . . . 23
7 Workflow diagram implementované aktualizace modelu pozadí 25
8 Vlevo zpracovávaný snímek videosekvence, vpravo maska popředí vytvořená algoritmem z kapitoly 4 27
9 Průměrný čas zpracování jednoho snímku videosekvence 1 28
10 Průměrný čas zpracování jednoho snímku videosekvence 2 29

Seznam výpisů zdrojového kódu

1 Struktura a pole pro ukládání dat modelu pozadí 23
2 Renormalizace vah distribucí pro aktuální pixel. Pokud nebyla nalezena shoda, je nově přidaná distribuce v modelu vynechána z renormalizace vah, protože to by její původní velmi malou hodnotu okamžitě katapultovalo nepřijatelně vysoko. Její hodnota je tedy rozdělena mezi zbývající distribuce tak, aby součet všech vah byl roven 1. 24
3 Funkce vracející normální rozložení pravděpodobnosti, upravená jako auxiliární funkce kernelu 26
4 Měření času vykonávání aktualizace modelu 28

1 Úvod

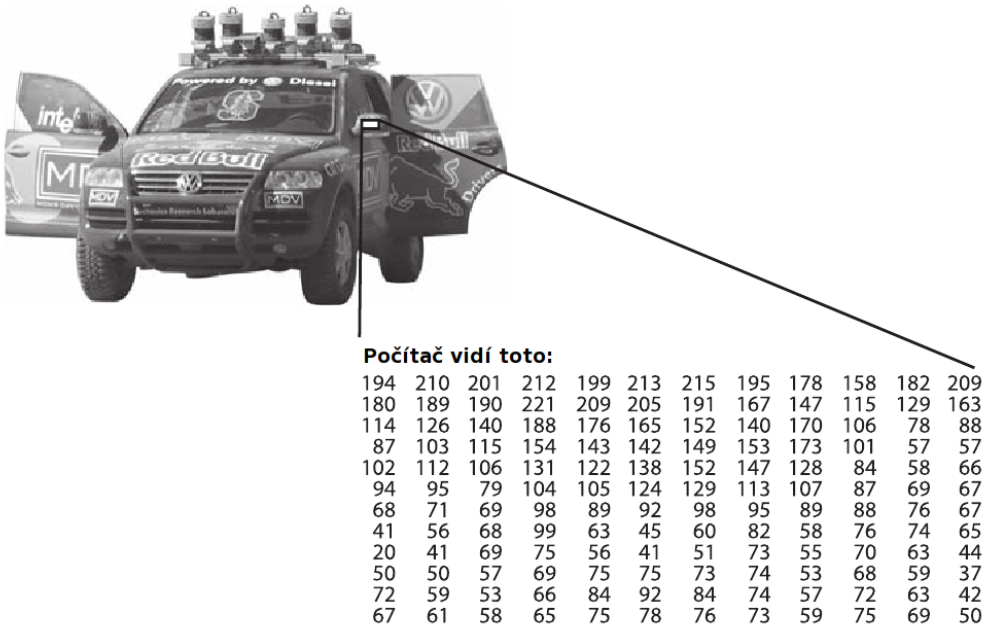
Detekce pohybu ve videosekvencích je velmi důležitým prvkem v nejrůznějších oblastech nasazení sledování videokamerami. Jako nejběžnější příklad můžeme uvést sledování a analýzu dopravního provozu na křižovatkách, dálnicích, parkovištích či jiných místech. V poslední době také nabývá na oblíbenosti a rozšířenosti sledování a rozpoznávání lidských gest při interakci se stroji. Detekce pohybu bývá nejčastěji prováděna odečtením pozadí, které je tvořeno buď referenčním snímkem nebo modelem pozadí. Každý jednotlivý snímek je pak od tohoto pozadí odečten a všechny body, jejichž rozdíl se výrazně liší, jsou považovány za pohyb. Je důležité, aby výsledek tohoto kroku byl co nejpřesnější, protože jakékoliv další zpracování zkreslených dat by mohlo vést k úplně jiným výsledkům, než bychom chtěli.

Každý algoritmus řešící problém detekce pohybu musí být schopný se správně vypořádat s několika nejčastějšími problémy. Uveďme si je na příkladu kamery, která snímá provoz na křižovatce. Algoritmus použitý v tomto případě musí být schopen se přizpůsobit různým denním podmínkám, jako jsou jiná osvětlení v noci a ve dne, s tím související různé úhly a délky stínů vrhané sledovanými objekty v různý čas dne a v neposlední řadě musí být nezávislý na počasí, které se také může libovolně měnit. Algoritmus musí také zvládnout správnou detekci stojících aut čekajících na zelenou, protože tato auta z pohledu algoritmu rychle splynou s pozadím díky své nepohyblivosti. K použití takového algoritmu v reálném čase nesmí být algoritmus náročný na výpočetní výkon a použitou paměť, ale i přesto si musí zachovat schopnost co nejpřesnější identifikace pohybujících se objektů.

Segmentování videosekvencí a následné zpracování je součástí větší skupiny procesů obecně nazývaných počítačové vidění. Počítačové vidění je široce rozprostřeno přes oblasti informatiky, optiky, matematiky, mechaniky, fyziky a z části i biologie a psychologie. Zjednodušeně se dá říci, že se jedná o vědu zabývající se programováním počítačů ke zpracování a především porozumění obrazových dat, jakými mohou být prosté obrázky, videosekvence, stereovizuální video (pohled ze dvou kamer současně), nebo vícerozměrná data z lékařských skenerů.

Porozumění obrazu počítačem je velice složitý proces. Člověk se svým biologickým viděním dokáže např. okamžitě rozpoznat auto na obrázku. Toto je však výsledkem složité práce lidského mozku, který rozloží vizuální signál na nespočet kanálů, které vedou nejrůznější informace do různých oblastí mozku. Mozek využívá širokou škálu asociativních informací ze všech možných senzorů a díky nim je schopen křížové asociace s poznatky, které jsme nabyli za dobu našeho pobytu na světě. Mozek skrz zpětnou vazbu řídí mechanické vlastnosti očí, jako je množství propouštěného světla skrz rohovku a upravuje příjem světla na povrchu sítnice.

Počítač oproti tomu přijme na vstupu pouze řadu čísel. Nemá žádný systém rozpoznávání tvarů, nedokáže automaticky ovládat zaostření obrazu, ani není schopen křížové asociace z letitých zkušeností. Na obrázku 1 vidíme auto, které má na straně řidiče zpětné zrcátko. Počítač však vidí pouze síť čísel, která je navíc z části zkreslená šumem způsobeným nedokonalostí záznamních přístrojů. Tato síť čísel nám dává velmi málo informací



Obrázek 1: Způsob, jakým vidí počítač

o vlastním obraze. Ale toto je vše, co „vidí“ počítač. Základním problémem je reprezentace trojrozměrného světa dvourozměrným obrazem. Neexistuje žádný způsob, jak z tohoto obrazu dostat zpět trojrozměrný signál. Tento 2D obraz může reprezentovat nekonečnou kombinaci 3D scén.

Oblast počítačového vidění se dá stále považovat za nevyvinutou. Díky různorodosti oborů, do kterých počítačové vidění zasahuje, neexistuje žádná standardní formulace „problému počítačového vidění “. Neexistuje ani formulace toho, jak by problémy počítačového vidění měly vůbec být řešeny. A tak místo toho vznikla spousta metod řešících nejrůznější dobře definované úlohy počítačového vidění. Tyto metody však byly často velmi úzce zaměřeny na danou úlohu a nebylo je tak možné použít pro širší aplikaci. Mnoho těchto metod je stále ve fázi výzkumu, ale jsou i takové, které se již uplatnily i v komerčních produktech. Jednou takovou metodou je právě segmentace popředí.

2 Oddělení popředí od pozadí

Existuje spousta algoritmů pro oddělení pozadí od popředí, většina z nich však postupuje podle jednoduchého schématu jako je na obrázku 2. Toto schéma tvoří čtyři základní kroky – preprocessing, modelování pozadí, detekce popředí a validace dat. Preprocessing zajišťuje přípravu obrazového snímku pro další použití v samotném algoritmu. To zahrnuje např. převod do srozumitelného formátu pro algoritmus, změnu parametrů videosnímku jako např. velikost, barevnost, jas apod. Takto připravený snímek pak přebírá proces modelování pozadí. Ten na základě vstupních hodnot počítá a aktualizuje současný model, který je statistickou reprezentací pozadí videa.

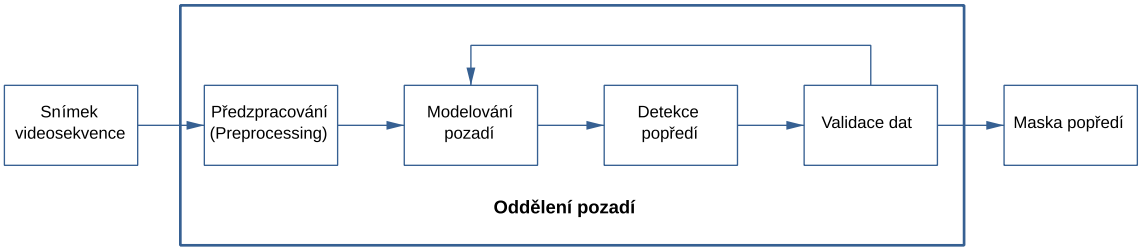
Modely se dělí na statické a dynamické. Statický model pozadí je vytvářen v tzv. učící fázi a po dobu trvání videosekvence již není obměňován. Z tohoto jasně vyplývají nevýhody – je potřeba získat dlouhou část videosekvence bez popředí pro učící část, model poté již není schopen reagovat na dlouhotrvající změny v pozadí (není obměňován) a také nedokáže reagovat na náhlé změny osvětlení scény. Dynamické modely naopak svůj model během videosekvence neustále obměňují. Jsou tak schopny reagovat na různé změny v pozadí scény. Nevýhodami těchto modelů je problém s nastavením parametrů pro rychlost adaptace na změny ve scéně – např. objekt, který se zastaví, splyne s pozadím, nebo naopak se pohybuje příliš rychle a tak v modelu vznikají duchové.

Pixely, které nezapadají do modelu pozadí, jsou pak označeny procesem detekce popředí jako samotné popředí. Takto označené pixely pak tvoří binární masku, tedy 1 = popředí, 0 = pozadí. Tím se získá hrubý odhad samotného popředí, ten je ale potřeba ještě ověřit. K tomu slouží krok validace dat. Zde se kontroluje, zda-li konkrétní body v masce skutečně odpovídají pohybujícím se objektům. Z masky se odstraní ty, které neodpovídají a tato maska se pak předá na výstup jako samotné popředí. V tomto kroku se používají výpočetně-náročné algoritmy, i přesto ale zde zůstává možnost real-time zpracování, protože algoritmy jsou použity pouze na malé množství bodů v masce. Pro každý z těchto kroků existuje několik různých návrhů, ty nejčastější budou popsány v následujících kapitolách.

2.1 Preprocessing

Tento krok by se dal rozdělit do dvou částí – globální a lokální preprocessing. Globální preprocessing umožňuje ušetřit výkon při samotném modelování pozadí, protože místo zpracování každého snímku zvlášť se dopředu zpracuje rovnou celá videosekvence. To je samozřejmě možné pouze v případě, kdy máme uloženou konečnou videosekvenci. Nelze takto zpracovat např. streamované video, zde je nutné použít lokální preprocessing. Ten je pak použit na každý snímek na vstupu zvlášť.

Zpracování může zahrnovat konverzi celého videa do jednoduššího formátu pro další použití. Každé dodatečné dekódování videa zbytečně přidává další výpočetní úlohy a tak snižuje výkon celého procesu. Většinou následuje vyhlazení videa, které má za úkol odstranit šum kamery nebo jiných přírodních ruchů zachycených kamerou, jako je déšť či sněžení. Velmi obvyklé je také zmenšení snímku nebo snížení počtu snímků za vteřinu (běžné video je kolem 24 snímků za vteřinu). Samotný algoritmus detekce popředí se dá



Obrázek 2: Proces segmentace pozadí

zrychlit odstraněním nezajímavých části pro konkrétní situaci. Pro křižovatku můžeme z videosnímku vyjmout budovy, chodníky či oblohu, pohyb v těchto částech nás nezajímá. Mezi dalšími operacemi je např. převedení snímku do odstínu šedi pro nižší náročnost a komplexnost algoritmu, ale často za cenu menší přesnosti. Snížení komplexnosti však bývá poměrně vysoké, protože se nemusí udržovat model pro každou barvu nezávisle.

2.2 Modelování pozadí

Základem každého algoritmu, oddělujícího popředí od pozadí, je modelování pozadí. Tomuto kroku byla věnována velká část výzkumu při vytváření modelu schopného odolat různým rušivým faktorům v pozadí, jako je např. změna počasí, množství světla, hýbající se větvičky stromů či šum kamery, ale i navzdory všem ruchům schopného správně detekovat pohyb.

Obecně můžeme modely rozdělit do dvou širších skupin – rekurzivní a nerekurzivní. Popsány budou pouze dynamické modely, tedy ty metody, jenž se dokážou snadno přizpůsobit změnám ve videosekvencích a nevyžadují inicializaci příliš náročnou na zdroje. Takovou inicializací může být např. načtení několika desítek vteřin videa dopředu pro vytvoření základního modelu. Tuto inicializaci využívají např. modely popsané vlastními charakteristickými obrázky (eigen-images).

2.2.1 Ne-rekurzivní techniky

Ne-rekurzivní algoritmy udržují v paměti určitý počet posledních snímků, na základě kterých se sestavuje model pozadí. Díky tomuto plovoucímu oknu jsou tyto algoritmy vysoce přizpůsobivé změnám v pozadí, protože k výpočtu se používají pouze snímky uložené v paměti a cokoliv bylo ve videu před nimi, se nebere v úvahu.

Na druhou stranu může toto způsobit problém např. v pomalém silničním provozu. Pokud bude plovoucí okno tak krátké, že zachytí auto pouze při čekání na zelenou na semaforu, stane se toto auto při posunu plovoucího okna součástí modelu pozadí. Počet uložených snímků v historii je proto potřeba nastavit adekvátně situaci, což může zrovna v případě pomalého provozu na cestách vést k velkým paměťovým nárokům. Používáme-li pevně přidělenou paměť, dá se tento problém částečně překonat snížením snímkovací frekvence videa.

2.2.1.1 Rozdíl snímků (Frame differencing) Tato velmi jednoduchá metoda používá k vytvoření modelu pozadí pouze předchozí snímek, I_{t-1}, který se jednoduše odečte od snímku aktuálního:

$$|I_t - I_{t-1}| > T,$$

kde I_t je intenzita snímku v čase t a T je pevně daný práh citlivosti. Zvolená hodnota prahu je pro tuto metodu zásadní, neboť je to jediný faktor, který ovlivňuje výsledek. Předpokladem pro tuto metodu je neustálý pohyb objektů při správné snímkovací frekvenci. Zjednodušeně lze říct, že odečtením zmizí z výsledné masky popředí vše, co se nehýbe. Tím nastává problém, když se sledovaný objekt zastaví. Pro tento model jakoby splyne s pozadím. Podobný problém nastává při větších, stejně zbarvených, pohybujících se plochách. Vnitřní pixely této plochy nebudou detekovány jako popředí, protože pixel v tom místě má stejnou hodnotu jako v předchozím snímku a tak dojde k jeho úplnému odečtení. Bez dodatečné úpravy algoritmu jsou tak detekovány pouze hrany objektů.

2.2.1.2 Lineární prediktivní filtr (Linear predictive filter) Tento algoritmus na úrovni pixelů používá Wienerův predikční filtr, který předpovídá na základě pravděpodobnosti, jaké hodnoty pixelů se objeví v dalším snímku. Jakýkoliv pixel, který se zásadně odchyluje od předpovězené hodnoty je označen jako popředí. Lineární predikce další hodnoty daného pixelu v čase t je

$$s_t = -\sum_{k=1}^{p} a_k s_t - k,$$

kde s_t je předpovězená hodnota pixelu, $s_t - k$ je předchozí hodnota pixelu a a_k je predikční koeficient, spočtený z vzorků kovariančních hodnot s_n. V případě $s_t > T$, kde T je zvolená odchylka, je pixel považován za popředí.

Problém nastává v modelu pozadí, když se objeví objekt označený jako popředí. Ten začne narušovat hodnoty použité jako historie pixelu. Z toho důvodu je dobré udržovat kromě historie předpovědí i historii skutečných hodnot. Pro každý nový pixel se tak vypočtou dvě predikce, jedna na základě skutečných hodnot a jedna na základě předpovězených hodnot. Pokud je alespoň jedna predikce menší, než zvolená odchylka, je pixel považován za pozadí. V každém snímku je ještě potřeba přepočítat predikční koeficient, aby byla zajištěna přizpůsobivost algoritmu na změny v pozadí. Je-li aktuální odchylka pixelu menší, než $1,1$ násobek předchozí odchylky stejného pixelu, je koeficient aktualizován, jinak je zachován starý koeficient.

Lineární prediktivní filtr je jednou z metod, které díky složitým výpočtům nelze použít v reálném čase.

2.2.1.3 Mediánový filtr (Median filter) Algoritmus mediánového filtru je další efektivní a velmi často používaná metoda na poli oddělování pozadí od popředí [6]. Vytvoření modelu pozadí spočívá ve výpočtu hodnoty mediánu pro každý pixel snímku ze všech snímků uložených ve vyrovnávací paměti (plovoucím okně). Poté kontrolujeme u každého pixelu jeho odchylku od vypočteného mediánu a podle nastaveného prahu označíme buď jako popředí nebo pozadí. Při správných parametrech dosahuje metoda velmi

dobrých výsledků při nízkých výpočetních nárocích, zato má však vysokou paměťovou náročnost díky snímkům uloženým v bufferu (Nx$framesize$).

2.2.1.4 Neparametrický model (Non-parametric model) Tento model počítá hustotu normálního rozložení (GDF) pro každý pixel na základě jeho historie. Elgammal [7] k tomu používá celou historii $I_t - L, I_t - L + 1, \ldots, I_t - 1$ ze které formuje neparametrický odhad GDF pixelu $f(I_t = u)$:

$$f(I_t = u) = \frac{1}{L} \sum_{i=t-L}^{t-1} K(u–I_i)$$

kde $K(\cdot)$ je funkce kernelu . Pokud zvolíme jako kernel funkci normálního rozložení $N(0, \Sigma)$, přičemž Σ reprezentuje šířku pásma kernelu, pak můžeme odhad počítat jako

$$f(I_t = u) = \frac{1}{L} \sum_{i=t-L}^{t-1} \frac{1}{(2\pi)^{\frac{d}{2}} |\Sigma|^{\frac{1}{2}}} e^{-\frac{1}{2}(u–I_i)^T \Sigma^{-1}(u–I_i)}$$

Použitím tohoto pravděpodobnostního odhadu pak oddělíme pozadí od popředí a to tak, že popředí je $f(I_t) < T$, kde T je zvolený práh citlivosti. Tím, že je použita Gaussovo normální rozložení jako K, se dá tento neparametrický model považovat za zobecněnou obdobu modelu mixtury gausiánů. Díky tomu je také odhad přesnější a při použití pouze části historie pixelu dokáže model vcelku rychle reagovat na změny v pozadí. Další výhodou použití GDF je multimodálnost celého modelu. To znamená, že se model dokáže vyrovnat s pozadím, ve kterém se například hýbe větvička stromu nebo se stavem, kdy se ostré kontrastní hrany díky nepatrnému pohybu kamery a šumu v obraze posunou v libovolném směru o pixel. Model podává velmi dobré výsledky, ale je také velmi náročný na výpočetní výkon.

2.2.2 Rekurzivní techniky

Oproti ne-rekurzivním technikám nepoužívají rekurzivní techniky vyrovnávací paměť snímků pro modelování pozadí, ale rekurzivně aktualizují model pozadí, který je odvozen ze všech předchozích snímků. I přesto, že se model aktualizuje pouze na základě právě zpracovávaného videosnímku, tak právě díky postupnému odvozování modelu ze všech snímků může mít i dávno minulý snímek špatný vliv na aktuální model. Výhodou oproti ne-rekurzivním metodám je tedy nižší paměťová náročnost, nevýhodou pak větší čas potřebný k „vyplavení“ chyb z modelu (při správném nastavení parametrů se však dá výrazně zkrátit).

2.2.2.1 Kalmanův filtr (Kalman filter) Kalmanův filtr vznikl už v roce 1960 a je považován za optimální rekurzivní algoritmus zpracovávající data. Jedním z důvodů jeho optimálnosti je schopnost zapracovat do výpočtu jakékoliv informace, které můžeme poskytnout. K výpočtu hodnot, které potřebujeme, používá veškeré naměřené hodnoty,

přičemž využívá znalost systému měření, statistické hodnoty ruchů v systému a odchylek a také jakékoliv informace o počátečních podmínkách výstupních hodnot. Tímto dokáže dát výsledek se statisticky minimální chybou. Například k zjištění rychlosti letadla lze použít Dopplerův radar, nebo údaje navigačního systému, nebo statický tlak a údaje o větru z příslušných zařízení. Kalmanův filtr může být postaven tak, aby dokázal zapracovat všechna tato data a díky znalosti dynamik jednotlivých systémů je schopný dát celkově nejlepší výslednou hodnotu rychlosti. Z mnoha verzí Kalmanova filtru, které existují, používá nejjednodušší verze pouze světelnost pixelu [8], Karmann a von Brandt používají světelnost a její dočasnou derivaci [4] a Koller, Weber a Malik pak používají světelnost a prostorovou derivaci [9]. V případě použití verze [4] je pozadí modelováno tímto vzorcem:

$$\begin{bmatrix} B_t \\ B_t' \end{bmatrix} = A \cdot \begin{bmatrix} B_{t-1} \\ B_{t-1}' \end{bmatrix} + K_t \cdot \left(I_t - H \cdot A \cdot \begin{bmatrix} B_{t-1} \\ B_{t-1}' \end{bmatrix} \right)$$

kde matice A udává dynamiku pozadí, matice K je Kalmanova zisková matice a H je měřící matice [measurment matrix]. Karmann a von Brandt ve své práci [4] uvádějí tyto výchozí hodnoty:

$$A = \begin{bmatrix} 1 & 0,7 \\ 0 & 0,7 \end{bmatrix}, H = \begin{bmatrix} 1 & 0 \end{bmatrix}$$

K_t se mění mezi pomalou – $alpha_1$ – a rychlou – $alpha_2 > alpha_1$ – adaptací podle toho, je-li $I_t - 1$ pixel náležící pozadí nebo popředí:

$$K_t = \begin{bmatrix} \alpha_1 \\ \alpha_1 \end{bmatrix} \text{ je-li } I_{t-1} \text{ popředí, jinak } \begin{bmatrix} \alpha_2 \\ \alpha_2 \end{bmatrix}$$

Známou nevýhodou je zanechávání dlouhých stop za pohybujícím se objektem.

2.2.2.2 Přibližný mediánový filtr (Approximated Median Filter)

V roce 1995 navrhli McFarlane a Schofield [10] algoritmus na sledování selat podle ne-rekurzivního mediánového filtru, ale s vlastnostmi rekurzivních technik. Neuchovává se tedy žádná paměť předchozích snímků, ale aktualizuje se jednosnímkový model pozadí. Později tato technika byla také použita pro modelování pozadí při sledování městského provozu.

Princip je velmi jednoduchý. Pokud na vstupu dostaneme pixel snímku, jehož hodnota I_t je větší než odpovídající hodnota v modelu, je tato hodnota modelu pozadí zvýšena o jedničku. Naopak je-li menší než hodnota v modelu, je tato hodnota o jedničku snížena. Model tedy směřuje ke stavu, kdy polovina všech pixelů je větší než hodnota v modelu a druhá polovina je menší. Dostáváme tedy medián.

Tato metoda je schopna dosáhnout velmi dobrých výsledků i přes svou jednoduchou implementaci. Plusem je také malá paměťová náročnost. Nevýhodou pak je velmi pomalá reakce na větší změny v pozadí.

2.2.2.3 Mixtura Gausiánů Stauffer a Grimson přišli s modelem, který místo jednoho gausiánu, jako v případě Kalmanova filtru, používá mixturu několika Gaussových distribucí najednou se systémem vah jednotlivých gausiánů – čím větší váha, tím pravděpodobněji náleží pozadí. Pro minimální funkčnost uvádí [1] alespoň 3 gausiány, u více než 6 gausiánů je pak zlepšení už velmi zanedbatelné. Algoritmus porovná hodnotu aktuálního pixelu I_t se všemi gausiány v modelu a najde takový gausián, do jehož standardní odchylky, $2,5\sigma$, hodnota pixelu zapadá. Parametry tohoto gausiánu jsou poté patřičně upraveny a jeho váha zvýšena. Distribuce jsou poté seřazeny podle koeficientu ω/ρ a následně jsou jako pozadí vybrány ty distribuce, jejichž součet vah je větší než zvolený práh T. Pokud aktuální pixel I_t byl přiřazen některé ze zbylých distribucí, je tento pixel označen jako popředí. Více o této metodě v kapitole 3.

2.3 Detekce popředí

Když máme sestaven model pozadí, který je ve většině metod reprezentován jedním obrázkem, stačí jednoduše odečíst tento vymodelovaný obrázek B_t od aktuálního snímku videosekvence I_t a výsledek porovnat se zvoleným prahem citlivosti T:

$$|I_t(x,y) – B_t(x,y)| > T.$$

Je-li tedy rozdíl větší, než zvolený práh T, je pixel o souřadnicích (x,y) považován za popředí. Jiný způsob detekce popředí je spočítáním normalizované statistiky:

$$\frac{|I_t(x,y) – B_t(x,y) – \mu_d|}{\sigma_d} > T_s,$$

kde μ_d je vrchol Gaussovy funkce a σ_d je standardní odchylka pro všechny souřadnice (x,y). Práh T není pevně dán, naopak se nastavuje s ohledem na situaci ve videosekvenci – některé scény vyžadují nižší citlivost, jako třeba scény s nízkým kontrastem, jiné naopak vyžadují citlivost vyšší. Fuentes a Velastin navrhují pro oddělení popředí použít spíše relativní rozdíl než absolutní [11]. Tím se zdůrazní kontrast v místech scény, která jsou ve stínu nebo jinak ztmavena oproti zbytku scény.

$$\frac{|I_t(x,y) – B_t(x,y)|}{B_t(x,y)} > T_c$$

Jiný přístup navrhuje použít dva prahy citlivosti, jeden pro „silnější“ pixely popředí, které jsou třeba uprostřed sledovaných objektů a jejich absolutní rozdíl s modelem pozadí je větší než tento práh T_h. Od těchto pixelů se pak počítá rozdíl všech okolních pixelů, který pro zařazení do popředí musí být větší než druhý, menší práh $T_l < T_h$.

2.4 Validace dat

Proces detekce popředí předá na vstup bitovou masku s pravděpodobným popředím. Ovšem data v masce nemusí být úplně správná, protože při detekci popředí se nebere v úvahu vztah sousedních pixelů, následkem čehož můžeme v bitové masce mít náhodný

šum. Tento problém se nejčastěji řeší kombinací morfologického filtru a shlukováním spojených komponent. Použitím morfologického filtru se zbavíme pixelů-sirotků, tedy osamocených pixelů, které jsou buď špatně označeny jako popředí nebo naopak jako pozadí. Morfologickým filtrem můžeme také spojit přilehlé regiony popředí, které odděluje několikapixelová „díra“.

Shlukováním spojených komponent pak můžeme sledovat konkrétní spojené regiony popředí a v případě, že jsou některé menší než by měly být (například sledujeme-li projíždějící auta a do záběru nám vletí pták), z masky se odstraní.

Dalším problémem je rychlost adaptace modelu pozadí na pohyby v záběru. Je-li adaptace pomalejší než pohyb, vznikají tzv. „duchové“, což jsou oblasti špatně detekovaných regionů popředí. Naopak je-li adaptace příliš rychlá, může detekce pomalejších objektů selhat. Několik autorů proto navrhuje používat více modelů s různými rychlostmi adaptace a periodicky porovnávat výsledek mezi těmito modely. Další možností, jak odstranit duchy z popředí, je spočítat jejich optický tok. To nám eliminuje nepohyblivé objekty, tedy různé stopy které za objektem zůstaly v modelu. Jedním z řešení je také sledovat scénu ze dvou kamer pod různými úhly a tuto informaci o úhlech použít pro výpočet hloubky. Na základě hloubky pak jsme schopni lépe rozpoznat popředí, neboť objekty v popředí jsou blíž než pozadí.

Posledním, v literatuře často zmiňovaným problémem, je drobný pohyb v pozadí, např. listí na stromech. Tento problém vcelku elegantně řeší metoda Mixtura Gausiánů, která také byla na tyto situace od počátku stavěna. Dodatečným morfologickým filtrováním pak masku popředí dočistíme.

3 Mixtura Gausiánů

Tento algoritmus formulovali v roce 1999 C. Stauffer a W. Grimson. Jejich původní implementace však měla několik nedostatků, proto později na základě této metody vzniklo několik dalších, vylepšených a dnes vcelku rozšířených metod. Tato adaptivní metoda používá k modelování pozadí mixturu několika normálních (Gaussových) distribucí. Každá tato distribuce představuje pravděpodobnost výskytu určité intenzity nebo barvy pixelu. Vrchol distribuce představuje intenzitu daného pixelu nebo hodnotu jedné barvy ze složek RGB. Jevy ve scéně jako kolísání jasu či šum kamery mohou dočasně posunout hodnotu pixelu mimo střed distribuce. Ke správnému zachycení tohoto pixelu je využita hodnota variance dané distribuce. Tato metoda předpokládá zpracování snímku na vstupu po jednotlivých pixelech (oproti zpracování celých oblastí) a také rozhodování o popředí nezávisle na předchozích snímcích (nepoužívají se tedy údaje z procesu validace dat ani údaje o sledování objektů).

Metoda je dost robustní, aby se dokázala vypořádat s náhlou změnou osvětlení scény, se sledováním objektů za překážkami v pohledu (např. sledování objektu mezi větvemi stromu), s pomalu se pohybujícími objekty a také s objekty, které se ve scéně náhle objeví nebo ze scény náhle zmizí. V tomto popisu se budeme zabývat algoritmem pouze pro monochromatickou video sekvenci.

3.1 Teorie

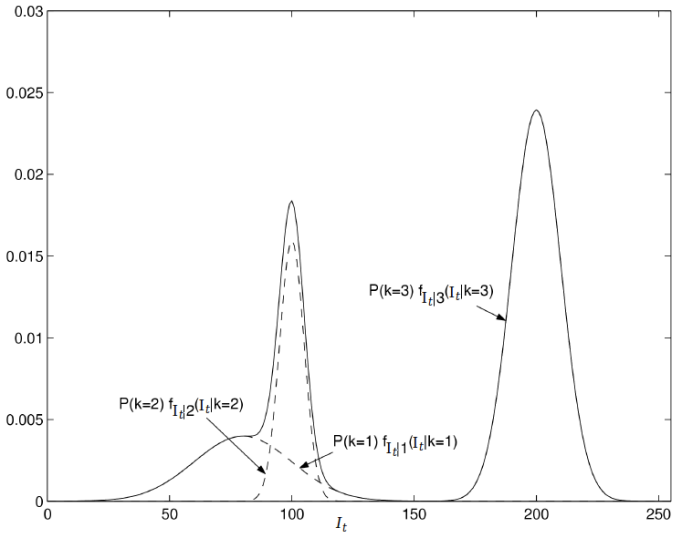
Každý objekt, který se dostane do pohledu daného pixelu, je reprezentován jedním stavem z množiny stavů $k \in 1, 2, \ldots K$, kde celkový počet stavů K je konstanta obvykle zvolená mezi 3 až 7. Část těchto K stavů odpovídá pozadí, zbytek pak tvoří popředí. Každá z těchto K distribucí je parametrizována vrcholem μ_k, variancí σ_k a dodatečným parametrem váhy gausiánu ω_k. Tento parametr představuje frekvenci, se kterou se daný stav doposud objevoval a také říká, jaká je (a priori) pravděpodobnost, že aktuální pixel I_t bude náležet právě tomuto stavu, přičemž platí

$$\sum_{k=1}^{K} \omega_k = 1. \tag{1}$$

Hodnota pixelu I_t je v případě monochromatického vstupního snímku skalár, v případě normalizovaného barevného prostoru je pixel reprezentován dvou-složkovým vektorem a je-li použit plně barevný snímek, je pixel tvořen tří-složkovým vektorem. Objekt, který je právě v pohledu pixelu, není znám a je pouze nepřímo odpozorován z hodnoty pixelu I_t. I kdyby objekt byl znám, stejně by musel být reprezentován jednou z K distribucí, protože hodnota I_t nebude díky šumu kamery či nepatrné změně osvětlení konstantní, ale bude se drobně vychylovat.

Pravděpodobnost, že na vstupu bude aktuální pixel je:

$$P(I_t) = \sum_{k=1}^{K} \omega_{k,t} * \eta(I_t, \mu_{k,t}, \Sigma_{k,t}), \tag{2}$$



Obrázek 3: Příklad rozložení pravděpodobnosti pixelu, rovnice (2).
$K = 3, I_t \in \{0, 1, ..., 255\}, \omega_k = \{0, 2, 0, 2, 0, 6\}, \mu_k = \{80, 100, 200\} a \sigma_k = \{20, 5, 10\}$.

kde K je počet distribucí modelu, $\omega_{k,t}$ je váha k-té distribuce v modelu v čase t, $\mu_{k,t}$ je vrchol k-té distribuce v čase t a $\Sigma_{k,t}$ je kovarianční matice k-té distribuce v čase t a η je normální rozložení pravděpodobnosti.

$$\eta(I_t, \mu_{k,t}, \Sigma_{k,t}) = \frac{1}{(2\pi)^{\frac{n}{2}} |\Sigma|^{\frac{1}{2}}} e^{-\frac{1}{2}(I_t - \mu_t)^T \Sigma^{-1} (I_t - \mu_t)}. \tag{3}$$

Obvykle se předpokládá, že rozměry I_t jsou na sobě nezávislé a tak je Σ_k pro jednoduchost inverze pouze diagonální. Původní práce Stauffera a Grimsona [1] předpokládá, že všechny odchylky σ_k jsou identické, z čehož plyne, že např. odchylky červené, zelené a modré barvy v barevném prostoru RGB jsou statisticky shodné. Kovarianční matici Σ_k lze tedy nahradit jednodušší diagonální maticí: $\Sigma_k = \sigma_k^2 I$. Je také možné tuto matici Σ_k nahradit jedním skalárem σ_k. V tomto lineárním barevném prostoru je to přijatelná aproximace, avšak v jiných prostorech, jako třeba HSV, je toto extrémním zjednodušením a proto nepoužitelné.

Distribuce v modelu jsou seřazeny sestupně podle pravděpodobnosti L_k, která je určena poměrem váhy distribuce ω_k a variance σ_k.

$$L_k = \omega_k / \sigma_k. \tag{4}$$

Nejprve je potřeba zjistit, která distribuce k pokrývá aktuální hodnotu pixelu I_t. Literatura nejčastěji používá dvě metody – původní metodu Stauffera a Grimsona [1] (i) a metodu s použitím Bayesova teorému [2] (ii).

(i) Nalezení shody $M_t \in [0, K]$ probíhá tak, že hodnota každého pixelu I_t je porovnána s vrcholem μ_k každé z K distribucí a je-li I_t ve standartní odchylce $2,5\sigma$ daného

vrcholu,

$$|\mu_{k,t} - I_{k,t}| < 2,5\sigma, \tag{5}$$

je tato distribuce vybrána jako „shoda“. Není-li I_t dostatečně blízko žádnému vrcholu distribuce, je $M_t = 0$. Velikost odchylky můžeme měnit bez vlivu na celkový výkon. To je velmi užitečné, když mají různé regiony různou intenzitu světla, algoritmus totiž můžeme přizpůsobit tak, aby hodnotu této odchylky měnil adekvátně podle situace. Jednotná odchylka pro celý snímek může vést ke „zmizení“ objektů, pokud se dostanou do stínu. Tento problém je blíže popsán v sekci 3.2 jako problém s dírami v objektu. Jedná se o stejný princip.

(ii) S aktuálním pixelem spočteme podmíněnou pravděpodobnost pro každou distribuci k v modelu podle Bayesova teorému:

$$P(k|I_t, \mu_{k,t}, \sigma_{k,t}, \omega_{k,t}) = \frac{P(k) f_{I_t|k}(I_t|k, \mu_{k,t}, \sigma_{k,t})}{f_{I_t}(I_t|\mu_{k,t}, \sigma_{k,t}, \omega_{k,t})} \tag{6}$$

a poté vybereme to k, které splňuje následující podmínku:

$$k = \arg\max_k \; P(k|I_t, \mu_{k,t}, \sigma_{k,t}, \omega_{k,t}) \tag{7}$$

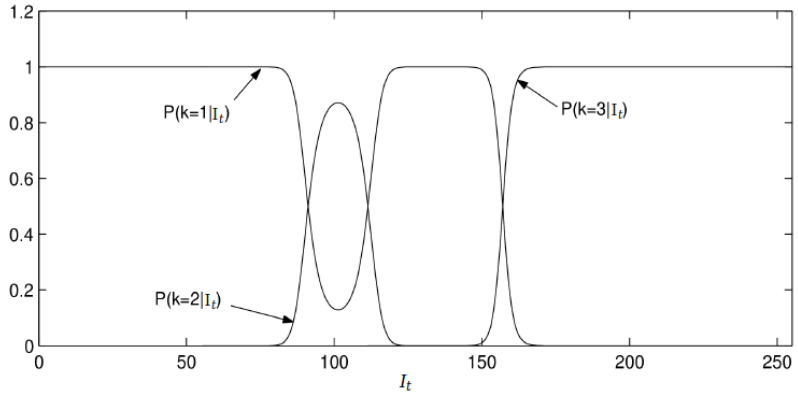
$$= \arg\max_k \; \omega_{k,t} f_{I_t|k}(I_t|k, \mu_{k,t}, \sigma_{k,t}). \tag{8}$$

Avšak zde dochází k problému, pokud aktuální pixel není pokryt žádnou z K distribucí modelu. Vybere se i přesto ta nejbližší. To je nejčastěji způsobeno objektem, který se nově objevil v pohledu pixelu. Rovnice (6) toto nebere v úvahu a nový objekt ve scéně by tak vyhodnotila špatně. Možné řešení je přidání další distribuce do modelu, který bude mít nízkou váhu, nedefinovaný vrchol a nekonečnou varianci. Tím dosáhneme stavu, kdy (6) vybere tuto distribuci, pokud žádná jiná neodpovídá.

Pokud není pro I_t nalezena vhodná distribuce ($M_t = 0$, nebo je podle (ii) vybrána distribuce $K + 1$), je nahrazena nejméně pravděpodobná distribuce $k_{Lmin} = \arg\min(L_k)$ novou distribucí, která má vrchol roven aktuální hodnotě pixelu $\mu_{k_{Lmin},t} = I_t$, velkou varianci a velmi malou váhu. Velkou výhodou je, že nahrazení stávající distribuce s nejmenší pravděpodobností výskytu nerozbije celý model. Barvy pozorované předtím v modelu zůstávají, než se samy stanou nejméně pravděpodobnou distribucí a budou nahrazeny jinými distribucemi. Získáváme tak výhodu pro objekty, které stály na místě dost dlouho na to, aby byly začleněny do pozadí. Když se znovu pohnou, distribuce, jenž byla označena za shodu předtím, než do pohledu pixelu vstoupil nový objekt, v modelu stále existuje se stejnými parametry, ale nižší váhou, avšak reprezentuje-li tato distribuce plně viditelný pixel, distribuce váhu rychle získá zpátky.

Je-li nově vytvořená distribuce způsobena dočasným objektem, který může být klidně šumem kamery, distribuce díky své malé váze model nijak nenaruší a časem z něj úplně vypadne, když se ve scéně objeví nový objekt.

Nově přidaná distribuce označuje pixel jako popředí, ale protože se nemusí jednat o skutečné popředí, je potřeba toto později ověřit.



Obrázek 4: A posteriori pravděpodobnosti, rovnice (6), vykresleny jako funkce I_t pro každé $k = 1, 2, 3$. Použity jsou stejné parametry jako v obr. 3

Byla-li nalezena shoda, jsou parametry této distribuce patřičně aktualizovány:

$$\omega_{k,t} = (1 - \alpha_t)\omega_{k,t-1} + \alpha_t P(k|I_t, \mu_k, \sigma_k), \tag{9}$$

kde α_t je parametr rychlosti učení se. Určuje tedy rychlost, s jakou se nehybný objekt stane součástí pozadí. Použijeme-li metodu (i) pro nalezení shody, definují Stauffer s Grimsonem v [1] možné zrychlení v podobě nahrazení poměrně náročného výpočtu pravděpodobnosti $P(k|I_t, \mu_k, \sigma_k)$, jednoduchou aproximací

$$A_{k,t} = \begin{cases} 1 & \text{distribuce označena jako shoda} \\ 0 & \text{ostatní případy} \end{cases} \tag{10}$$

$$\approx P(k|I_t, \mu_k, \sigma_k). \tag{11}$$

To vychází z pozorování, že $P(k|I_t, \mu_k, \sigma_k)$ je pro většinu hodnot I_t rovno buď 0, nebo 1 a pouze pro jedno k je velmi blízko 1, viz graf na obrázku 4.

Ostatní parametry distribuce jsou také aktualizovány:

$$\mu_{k,t} = (1 - \rho)\mu_{k,t-1} + \rho I_t, \tag{12}$$

$$\sigma^2_{k,t} = (1 - \rho)\sigma^2_{k,t-1} + \rho|I_t - \mu_{k,t-1}|^2, \tag{13}$$

kde

$$\rho = \frac{\alpha_t P(k|I_t, \mu_{k,t}, \sigma_{k,t}, \omega_{k,t})}{\omega_{k,t}}. \tag{14}$$

Pokud je nalezeno více než jedna shoda, je vybrána ta distribuce, která má nejvyšší poměr $\omega_{k,t}/\sigma_{k,t}$. Parametry ostatních distribucí zůstávají stejné, mění se pouze váhy, které je potřeba renormalizovat, neboť platí (1). Dalším krokem je klasifikace popředí a pozadí. Po aktualizaci parametrů se distribuce seřadí podle hodnoty $\omega_{k,t}/\sigma_{k,t}$, kde nejpravděpodobnější distribuce pozadí jsou nahoře, zatímco méně pravděpodobné transientní

distribuce se drží vespod. Poté vybereme prvních B distribucí jako reprezentaci pozadí

$$B = \arg\min_b (\sum_{k=1}^{K} \omega_k > T), \tag{15}$$

kde T je uživatelem zvolený práh citlivosti, vyjadřující jak velká část dat je minimálně potřeba, aby se mohlo jednat o pozadí. Toto vybere ty „nejlepší“ distribuce, dokud není dosaženo určité části dat, T. Zvolíme-li T jako malou hodnotu, je model pozadí většinou unimodální, používá se tedy pouze jedna distribuce. Použitím pouze nejpravděpodobnější distribuce můžeme ušetřit výpočetní čas.

Naopak je-li hodnota T vyšší, je možné do modelu pozadí zařadit více distribucí reprezentujících více barev, způsobených např. pohybem listí na stromě, vlající vlajkou apod. Takto se vyhneme falešné detekci např. právě těch listů na stromě jako popředí. Pokud je $B < K$ a pokud byla nalezena shoda právě ve zbývajících distribucích – $B < k_m \leq K$, je pixel reprezentován touto distribucí označen v masce popředí jako skutečné popředí.

Novozelanďané Power a Schoonees pak přišli s alternativní aproximací místo nahrazení (11) ve (14) [2]. Navrženou změnou bylo nahradit (11)

$$\rho_{k,t} \approx \begin{cases} \frac{\alpha_t}{\omega_{k,t}} & \text{distribuce označena jako shoda} \\ 0 & \text{ostatní případy} \end{cases} . \tag{16}$$

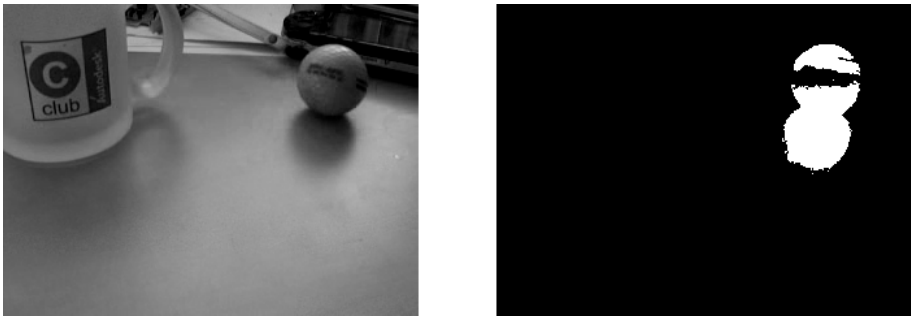
Takto se úplně vyhneme výpočtu $P(k|I_t, \mu_{k,t}, \sigma_{k,t}, \omega_{k,t})$ pouze za cenu dělení $\omega_{k,t}$. [2] uvádí, že se jedná o rychlejší a logičtější použití aproximace, než je (10)

3.2 Dodatečné rozšíření algoritmu

Pro praktické nasazení této metody v praxi je potřeba provést několik dodatečných úprav vygenerované masky popředí. Ve své základní formě totiž metoda občas chybně detekuje šum v obraze jako pohybující se popředí. K odstranění těchto bodů lze použít morfologický filtr, jak již bylo zmíněno v kapitole 2.4.

Dalším velmi častým jevem jsou díry uvnitř detekovaných objektů. Tento jev ale není chybou samotného algoritmu. Uvažme případ, kdy máme bíle auto s tmavými skly projíždějící místem ve stínu. Nechť hodnota I_t pozadí ve stínu je rovna 24. Když do pohledu libovolného pixelu uvnitř této zastíněné oblasti vjede ono bíle auto, bude hodnota I_t velmi vysoká, pro názornost uveďme 200. Toto bude jasně detekováno jako popředí. Díky pohybu vozu se ale zanedlouho dostane do pohledu tohoto pixelu tmavé sklo vozu, jehož hodnota I_t ve stínu nechť je 21. Jakmile algoritmus začne tento pixel zpracovávat, přiřadí ho podle (i) nebo (ii) distribuci reprezentující pozadí. Tím nám vznikla v objektu díra. Proto je potřeba masku zpracovat algoritmem, který díry uvnitř objektů vyplní.

Tento postup však může omylem vytvořit nové nežádoucí objekty. S tím související problém detekce falešného popředí lze vyřešit použitím filtru, který prozkoumá rozměry a tvar detekovaného objektu a pokud nevyhovuje zadaným kritériím, je z masky celý



Obrázek 5: Příklad vzniklých děr v masce popředí, vlevo zpracovávaný snímek videosekvence, vpravo maska popředí vytvořená algoritmem z kapitoly 4

objekt odstraněn. Toto však v mnoha aplikacích detekce pohybu není úplně triviální problém.

Obvyklou anomálií také bývá náhlá změna množství světla v obraze. To se může stát např. při průjezdu velmi světlého auta v blízkosti kamery, nebo při rozsvícení světla v tmavé místnosti, kdy kameře chvíli trvá než se přizpůsobí. Touto změnou je téměř okamžitě většina scény detekována jako popředí, neboť místo očekávaných původních nízkých hodnot I_t přišly na vstup hodnoty velmi vysoké. V tomto případě by mohlo pomoci samostatné sledování množství světla a v případě změny vynásobení všech parametrů distribucí koeficientem vyjadřujícím velikost změny osvětlení.

4 Implementace

Algoritmus mixtury gausiánů je implementován v jazyce C s použitím knihovny OpenCV verze 2.0. OpenCV byla zkompilována s podporou FFmpeg pro práci s videem a s podporou GTK pro práci s GUI. Pro OpenCL byla využita implementace ATI Stream 2.01. Více o použitých technologiích níže.

Tato implementace je založena na rozšířené a detailněji specifikované metodě popsané Powerem a Schooneesem v [2]. Aproximace nejsou využity, neboť samotná metoda byla navržena před 11 lety a dnešní výkon běžných počítačů je pro real-time zpracování videa o velikosti 320x240 pixelů dostačující.

Vývojovým a testovacím prostředím byl TabletPC hp tx2500z, osazen procesorem AMD Turion 64 X2, 3 GB RAM a integrovanou grafickou kartou ATI Radeon HD 3200. Operační systém byl použit GNU/Linux, distribuce Fedora 12 x86_64. Veškeré použité knihovny tedy byly 64-bitové.

Z důvodu použití distribuční knihovny FFmpeg, která je kvůli patentovému zatížení ochuzena o některé kodeky, jejichž patentová situace není zcela jasná nebo není v souladu s patentovou politikou distribuce a rovněž z důvodu možné ztráty kvality při převádění do jiného ztrátového formátu, byly testovací video sekvence převedeny do bezztrátového a bezkompresního formátu (rawi420). Vedlejším efektem pak je urychlení algoritmu, neboť není potřeba video dekomprimovat a dekódovat. Ovšem tento rozdíl byl víceméně zanedbatelný. Výkon se pohyboval okolo 4 snímků za vteřinu.

4.1 Použité technologie

4.1.1 OpenCV

OpenCV (Open Computer Vision) je svobodná multiplatformní knihovna pro počítačové vidění, původně vyvinuta firmou Intel. Specializuje se především na zpracování obrazu v reálném čase. Knihovna je psána převážně v jazyce C, části jsou napsány i v C++ a je nezávislá na hardwaru či použitém systému. Aktivně se pracuje také na rozhraních pro Python, Ruby, Matlab a několik dalších jazyků. Obsahuje přes 500 optimalizovaných algoritmů připravených k okamžitému použití z různých oblastí zpracování obrazu, jako např. kontrola produktů při výrobě, zpracování obrazu ve zdravotnictví či v bezpečnostním inženýrství, stereo vize, kalibrace kamery a také pokrývají část robotiky. Protože je počítačové vidění často spojováno se strojovým učením, obsahuje OpenCV také plnou knihovnu pro podporu strojového učení - Machine Learning Library (MLL). Tato knihovna se zaměřuje především na statistické rozpoznávání vzorů a klastrování.

Vývoj zahájil v roce 1999 Intel jako iniciativu o větší pokrok aplikací výpočetně náročných na CPU. Vývoj byl součástí série projektů, mezi které patřil ray-tracing v reálném čase nebo 3D obrazové stěny. Kromě Intel's Performance Library týmu se do vývoje zapojili i optimalizační experti z ruského Intelu. Intel definoval svojí snahu jako „konec znovuobjevování kola“. Jedním z cílů bylo také vytvořit jednotnou platformu, na které mohou vývojáři okamžitě začít stavět. Díky přenositelnosti knihovny mezi platformami

a kódu optimalizovanému na výkon a poskytovanému zdarma, chtěl Intel také urychlit nástup komerčních aplikací počítačového vidění.

První alfa verze byla vydána v lednu roku 2000, verze 1.0 pak až v roce 2006. Od poloviny roku 2008 přešla OpenCV knihovna pod záštitu korporace Willow Garage, specializující se především v oblasti robotiky. Vývoj se po přebrání opět aktivně rozběhl a ještě později toho roku byla vydána verze 1.1. V říjnu roku 2009 byla uvolněna verze 2.0, která přinesla především zvýšení výkonu stávajících funkcí, několik nových funkcí a také několik větších změn v rozhraní pro C++.

Dnes se OpenCV aktivně využívá pro skládání satelitních map, redukce šumu zdravotnických obrazů, analýzu objektů, bezpečnostní systémy a systémy pro detekci narušení, systémy pro automatické monitorování, inspekční systémy při výrobě a vojenské aplikace, jako je řízení bezpilotních letadel, pozemních a podvodních strojů.

Knihovna je uvolněna pod BSD licencí, uživatelé tedy nejsou nuceni při použití knihovny nebo její části dávat k dispozici svůj zdrojový kód, ani vracet zpět do projektu provedené změny knihovny. Díky tomuto můžeme mezi komerčními subjekty využívajícími OpenCV najít společnosti jako Intel, IBM, SONY, Siemens, Microsoft nebo Google. Využívá jí také několik výzkumných center, jako MIT, Stanford, CMU, Cambridge a INRIA.

OpenCV také implementuje algoritmus pro segmentaci videa, založený na vylepšené metodě Mixtury gausiánů, viz [3]. Tato práce však využívá OpenCV pouze pro práci s jednotlivými video snímky - načtení, převedení do odstínu šedi a také poskytnutí přímého přístupu k obrazovým datům jednotlivých snímků. Dále je využito API pro zobrazování obsahu, pomocí kterého je zobrazen vždy aktuální snímek video sekvence a odpovídající maska popředí. Volbou SHOW_DISTRIBUTIONS = 1 ve zdrojovém kódu je možné sledovat také stav distribucí pro konkrétní pixel (nastaven volbami WATCH_PIXEL_X a WATCH_PIXEL_Y).

4.1.2 OpenCL

OpenCL (Open Computing Language) je podobně jako OpenCV také multiplatformní knihovna, která umožňuje spouštění aplikací na různých typech procesorů, především pak CPU a GPU. OpenCL tedy zpřístupňuje výkon grafického procesoru negrafickým výpočtům. Aplikaci je dostupná také paměť GPU, která obvykle bývá rychlejší, než paměť systémová.

OpenCL definuje vlastní jazyk pro psaní kernelů, založen na standardu C99. Jazyk je tedy velmi podobný jazyku C a implementuje velké množství jeho vlastností, nedokáže však třeba rekurzivní volání funkcí. Kernelem je nazývána ta část kódu, která se spouští na určeném zařízení. Knihovna nabízí paralelní výpočty založené na úkolovém a datovém paralelismu.

Vývoj OpenCL započala společnost Apple, která po prvotním návrhu přizvala ke spolupráci techniky z firem AMD, IBM, Intel a Nvidia. Návrh specifikace byl poté svěřen do rukou konsorcia Khronos, které mimo jiné udržuje analogicky podobné standardy OpenGL pro 3D grafiku a OpenAL pro zvuk. Pracovní skupina uvnitř Khronosu začala okamžitě spolu s reprezentanty z CPU, GPU a embedded-procesorových výrobců dokon-

čovat technické detaily specifikace. První verze, OpenCL 1.0, byla pro širokou veřejnost vydána 8. prosince 2008.

K dnešnímu dni existuje několik implementací standardu OpenCL. AMD poskytuje zdarma uzavřený framework ATI Stream, jehož součástí je implementace OpenCL 1.0, kterou je možné provozovat jak na CPU, tak GPU. Apple zaintegroval tuto specifikaci do svého operačního systému Mac OS X 10.6 „Snow Leopard“. 30. října 2009 vydala IBM implementaci OpenCL 1.0 pod názvem OpenCL Development Kit for Linux on Power. Tato implementace je určena pro Power architekturu procesorů, která je používána hlavně v superpočítačích a je dostupná zatím pouze pro operační systém Linux. Nvidia vyvíjí vlastní výpočetní architekturu zvanou CUDA (Compute Unified Device Architecture). Avšak podporuje na svých zařízeních i OpenCL, k němuž vydala 26. listopadu 2009 ovladače v rámci svého SDK pro vývoj GPU akcelerovaných aplikací.

Tato práce využívá OpenCL paralelizace pro aktualizaci modelu pozadí. Každý pixel je tak aktualizován vlastním procesem výpočetního kernelu.

4.1.3 FFmpeg

FFmpeg je kolekce svobodných open-source knihoven a programů pro práci s multimediálním obsahem, především s videem a audiem. Tento poměrně velký projekt obsahuje na stovku nejrůznějších kodeků, mezi kterými je celá generace MPEG video kodeků (MPEG-1 Video, MPEG-2 Video, MPEG-4 Visual, MPEG-4 AVC) a MPEG audio kodeků (MP2, MP3, AAC, MPEG-4 ALS), kodedků Windows Media video a audio, sada kodeků pro práci s Real Player audio a video, QuickTime audio a video a také sada kodeků spojených s Adobe Flash videem. Zvládá také všechny známé video formáty, jako např. ASF, AVI, FLV nebo Matroska a vypořádá se i s nejrůznějšími protokoly, HTTP, RTP, TCP, UDP a GOPHER je pouze část všech podporovaných protokolů.

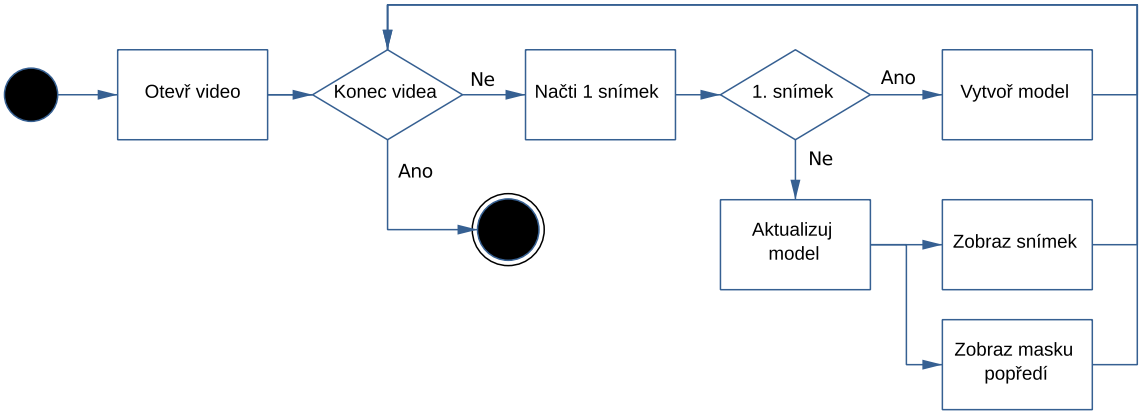
Na tomto projektu staví celá řada dalších projektů, jako jsou nejpoužívanější video přehrávač operačního systému Linux MPlayer, další velmi populární přehrávač multimédií VLC, internetový prohlížeč Google Chrome a v neposlední řadě využívá možnosti projektu FFmpeg také knihovna OpenCV.

V této práci je FFmpeg použit jako backend OpenCV pro dekódování a načítání video sekvencí.

4.2 Implementace

Nejdříve byla naprogramována implementace pouze s použitím OpenCV pro jednoduché ověření správné funkčnosti algoritmu. Zjednodušený postup algoritmu lze vidět na obrázku 6.

Na začátku bylo potřeba určit počáteční hodnoty distribucí a hodnoty parametrů ovlivňujících algoritmus. Po delším testování byly zvoleny hodnoty, které jsou uvedené v tabulce 1. Tyto hodnoty podávají velmi dobré výsledky ve video sekvencích, kde je prvních pár snímků vidět pouze pozadí. Uspokojivých výsledků bylo s těmito parametry dosaženo i v takových video sekvencích, kde je již od prvního snímku viditelný objekt



Obrázek 6: Workflow diagram implementovaného algoritmu Mixtury Gausiánů

K	4
α_{init}	0,1
T	0,75
ω_{init}	0,85
σ_{init}	15
μ_{init}	I_t

Tabulka 1: Počáteční parametry algoritmu

náležící popředí, což způsobí dočasně falešnou detekci popředí v místě, ze kterého se daný objekt přemístil.

Pro ukládání hodnot jednotlivých distribucí pro každý pixel modelu pozadí, slouží struktura `distributionValues`, která je pak použita jako typ proměnné v dvourozměrném poli `bgData`, jenž uchovává informace o všech distribucích celého snímku. Toto 2D pole má první rozměr o velikosti `šířka x výška` snímku a druhý rozměr je K+1. To proto, že je implementována varianta nalezení odpovídající distribuce Bayesovou větou (5) podle [2]. Do modelu je tedy potřeba přidat jednu „vyplňovací“ distribuci, která bude vybrána v případě, že se nepodařilo nalézt shodu.

```
typedef struct distributionValues
{
    float weight;
    float variance;
    float mean;
    float coef;
} distributionValues ;

distributionValues (*bgData)[K+1] = NULL;
```

Výpis 1: Struktura a pole pro ukládání dat modelu pozadí

Každý snímek videa je před samotným zpracováním nejdříve převeden do odstínů šedi pomocí OpenCV metody `cvCvtColor()`. Po převedení prvního snímku je prove-

dena inicializace modelu a vytvoření prvních distribucí. Všechny distribuce mají nastavený vrchol na aktuální hodnotu pixelu I_t a velikost variance na 15. První distribuce má nastavenou velmi vysokou váhu, $0,85$, ostatní pak mají váhu nastavenou na $(1-0,85)/(K-1)$. Distribuce $K+1$ je pak inicializována na velmi malou váhu, velmi vysokou varianci[1] a vrchol distribuce byl položen doprostřed rozsahu intenzity pixelu, tedy 128.

S každým dalším snímkem je provedena aktualizace celého modelu. Algoritmus projede snímek pixel po pixelu a pro každý pixel hledá shodu v podobě distribuce, která svým rozložením pokrývá jeho hodnotu. Je-li tato distribuce nalezena, jsou patřičně aktualizovány její hodnoty. Není-li nalezena shoda, je vybrána distribuce s nejnižší váhou a tato je nahrazena novou distribucí, která má vrchol roven aktuální hodnotě pixelu, variance je 15 a váha nové distribuce je $0,05$. Tato nově vytvořená distribuce je pak označena jako shoda.

Následně je provedena renormalizace vah, kterou je možné vidět na výpise 2. Po ní je provedeno seřazení distribucí vestavěným Quicksort algoritmem podle parametru `distributionValues.coef`, který uchovává podíl $\frac{\omega}{\sigma}$ a je přepočítáván vždy při renormalizaci vah.

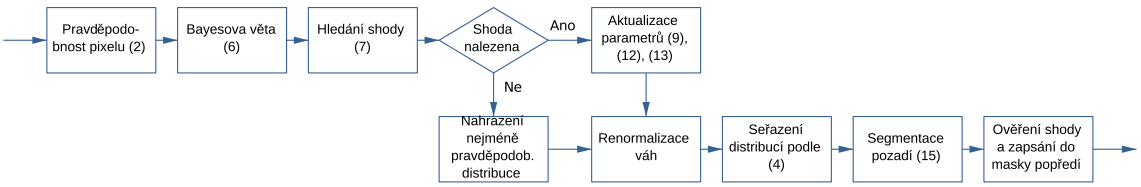
```
float gaussian_weight_sum = 0.0f;
float nonmatched_gauss_weight = 0.0f;

for(k=0; k<K; k++) {
    gaussian_weight_sum += bgData[n][k].weight;
}

if (matchFound == 0) {
    nonmatched_gauss_weight = (bgData[n][matchedK].weight / gaussian_weight_sum) / (K−1);
}

for(k=0; k<K; k++) {
    if (k != matchedK && matchFound == 0) {
   bgData[n][k].weight = ( bgData[n][k].weight / gaussian_weight_sum ) +
        nonmatched_gauss_weight;
   }
    else if (matchFound == 1) bgData[n][k].weight = bgData[n][k].weight / gaussian_weight_sum;
    bgData[n][k].coef = bgData[n][k].weight / bgData[n][k].variance;
}
```

Výpis 2: Renormalizace vah distribucí pro aktuální pixel. Pokud nebyla nalezena shoda, je nově přidaná distribuce v modelu vynechána z renormalizace vah, protože to by její původní velmi malou hodnotu okamžitě katapultovalo nepřijatelně vysoko. Její hodnota je tedy rozdělena mezi zbývající distribuce tak, aby součet všech vah byl roven 1.

Posledním krokem je vybrání distribucí, jejichž součet vah je větší, než zvolený práh T. Quicksort seřadí distribuce sestupně[2], je tedy nutné projít distribuce daného pixelu „odzadu“. Zbylé distribuce jsou potencionální reprezentace pixelu náležícího popředí.

[1] [2] uvádí zvolit nekonečno, avšak z praktických důvodu plně postačila hodnota $9999,99$

[2] řazení v Quicksortu záleží na implementaci porovnávací funkce, avšak při pokusu o opačné řazení docházelo k problémům, proto bylo ponecháno řazení sestupně



Obrázek 7: Workflow diagram implementované aktualizace modelu pozadí

Pokud byla některá z těchto distribucí označena jako shoda a není-li její váha příliš vysoká, je tento bod označen v masce popředí jako skutečné popředí.

Celý proces je možné vidět na obrázku 7.

4.2.1 OpenCL implementace

OpenCL implementace algoritmu mixtury gausiánů je z důvodů nedostupnosti podporovaného GPU napsaná pro běh na CPU. Avšak pro spuštění kernelu na GPU je potřebných pouze pár drobných změn ve způsobu inicializace a adresování paměti. Aktuálně je totiž použita hlavní systémová paměť'RAM, zpracování na GPU však vyžaduje použití paměti přímo na zařízení.

Před spuštěním vlastního kódu je potřeba provést inicializaci zařízení a alokovat místo v paměti pro data. OpenCL umožňuje pro své výpočty použít několik druhů zařízení - CPU, GPU, speciální OpenCL akcelerátory, nebo kombinaci všeho dostupného. Paměť se pro výpočetní zařízení alokuje skrz speciálně vytvořené objekty, tzv. buffery, které se přímo předávají kernelu. Paměť a data lze těmto bufferům alokovat dvěma způsoby. Buďto vytvoříme v paměti úplně nový region pro data, kde skrz volání OpenCL data nakopírujeme, nebo použijeme již existující alokovaná data v paměti a OpenCL na ně předáme pouze odkaz. Tato implementace používá druhou metodu, data jsou tedy připravena v paměti, vytvoří se z nich buffer s ukazatelem na alokovanou paměť a ten je poté předán kernelu.

Kernel je v podobě zdrojového kódu uložen v externím souboru. Při spuštění je ze souboru načten a JIT kompilátorem, který je součástí OpenCL, je za běhu zkompilován. V případě syntaktických chyb kompilátor vypíše příslušné řádky kódu spolu s chybou. Kernel je také možné zkompilovat trvale do binární podoby a načítat tak připravenou binárku.

Po provedení všech nezbytných inicializací je načten první snímek videa a stejně jako v předchozí implementaci, je metodou `cvCvtColor` převeden na odstíny šedi a na základě tohoto snímku vytvořen prvotní model pozadí. S prvním snímkem získáváme rozměry videosekvence a podle těch nastavíme celkový počet kernelů, které se budou spouštět, pro každý pixel jeden. Každý kernel je voláním metody `clEnqueueNDRangeKernel` zařazen do fronty ke zpracování. Celkem tedy např. pro videosekvenci s rozměry 320x240 je do fronty zařazeno 76 800 vláken kernelu pro každý jeden snímek.

Pro data, ze kterých jsou vytvořeny buffery a které potřebujeme změnit mimo kernel, je potřeba po skončení zpracování každého snímku říct OpenCL, že se data budou

měnit. Metoda `clEnqueueMapBuffer` zařídí, že OpenCL zapíše veškeré změny v odkazovaných datech zpět do bufferu. Před zařazením kernelu do fronty je nutné toto „monitorování“ ukončit zavoláním `clEnqueueUnmapMemObject`, tím se příslušný buffer opět uzavře pro změny a připraví k použití. Příkladem této manipulace je snímek videosekvence, který se po každé iteraci aktualizuje načtením nového snímku.

Samotný proces aktualizace modelu pozadí je téměř stejný, jako v implementaci bez OpenCL. Bylo potřeba pouze upravit způsob indexování přístupu k prvkům pole, ve kterém je model uložen, protože OpenCL podporuje pouze jednorozměrné pole. Každý spuštěný kernel zná své vlastní pořadové číslo v rámci celé skupiny. Díky tomu, že OpenCV ukládá obrazová data také lineárně do jednorozměrného pole, je možné jako index v tomto poli využit právě pořadové číslo kernelu, tím dosáhneme toho, že právě jeden kernel bude zpracovávat právě jeden pixel. Analogicky je pak toto indexování využito při přístupu k datům modelu pozadí. Index `[n][k]` tak nahradila univerzálnější konstrukce `[(K + 1) * n + k]`.

OpenCL kernel také nezná konstanty deklarované stylem `#define`. Proto musely být všechny potřebné konstanty nahrazeny prostými proměnnými. Pro rozlišení zůstala alespoň zachována konvence pojmenovávání konstant velkými písmeny.

OpenCL umožňuje použít i externí metody v rámci kernelu, které je možné z kernelu volat. Dokumentace ATI Stream však nedoporučuje používat funkce vracející hodnoty, místo toho doporučuje raději předat ukazatel libovolné proměnné jako jeden z parametrů a do této proměnné výsledek zapsat. Takto upravena byla funkce `gaussf()`, která počítá normální rozložení pravděpodobnosti.

```
/*
 * Parametry: x – hodnota aktuálně zpracovávaného pixelu
 *     mu – vrchol distribuce
 *     sigma – variance distribuce
 *     res – ukazatel proměnné pro výsledek
 */
void pdf(float x, float mu, float sigma, float * res) {
    float m = 1.0f / ( pow( (2.0f * 3.14159265f * sigma), 1.0f / 2.0f) );
    float m2 = exp( -( pow( x-mu, 2.0f ) ) / (2.0f * pow( sigma, 2.0f ) ) );
    float temp = m*m2;
    *res = temp;
}
```

Výpis 3: Funkce vracející normální rozložení pravděpodobnosti, upravená jako auxiliární funkce kernelu

5 Testování

Původním záměrem bylo zjistit rozdíl mezi výkonem základní implementace spouštěné na CPU a mezi OpenCL implementací spouštěné na GPU. Avšak z důvodu nedostupnosti stroje vybaveného OpenCL podporovaným GPU, na kterém by mohla být OpenCL implementace spuštěna a řádně otestována, je varianta implementace s OpenCL také testována na CPU.

Testovací sestavu poháněl dvoujádrový procesor Intel Core 2 Duo s frekvencí 2,13 GHz každého jádra. Systém pracoval se 4 GB RAM DDR3. Operačním systémem byl 64 bitový GNU/Linux, distribuce Ubuntu 9.10. Stejně jako pro vývoj, bylo pro testování OpenCL použito ATI Stream SDK.

V testech byl porovnán výkon základní jednoprocesové implementace a paralelního zpracování OpenCL. Testy byly provedeny na dvou videosekvencích:

1. záznam o 1000 snímcích a rozlišení 320x240 ve formátu nekomprimovaného RAW
2. záznam o 200 snímcích a rozlišení 768x576 ve formátu MPEG-ES

Obě videosekvence obsahují stejný záznam provozu na křižovatce. Videosekvence 1 byla testována pro zjištění, zda-li OpenCL na CPU nebude na tak malém rozlišení podávat naopak horší výsledky než implementace algoritmu bez OpenCL.

Obrázek 8: Vlevo zpracovávaný snímek videosekvence, vpravo maska popředí vytvořená algoritmem z kapitoly 4

Měřen byl čas vykonávání aktualizace modelu pro každý snímek a z těchto časů poté vypočten aritmetický průměr. V následujících grafech jsou zobrazeny právě tyto vypočtené průměry. Pro měření času byla použita standardní funkce `clock()` (definována v `time.h`) před a po volání metody pro aktualizaci modelu, jak je možné vidět na výpise 4. Metoda `clock()` vrací počet taktů vnitřních hodin od spuštění programu. Proto je potřeba tuto hodnotu patřičně převést na milisekundy vydělením makrem `CLOCKS_PER_SEC`, které zajistí dosazení počtu taktů za vteřinu aktuálního procesoru. Pro výpočet aritmetického průměru je použit (počet snímků -1), to proto, že první snímek je použit pro tvorbu modelu.

```
clock_t start, update;
...
start = clock();
updateModel(pGSFrame);
update = clock();
lTime = (update - start);
timeSum += lTime * 1000.0 / CLOCKS_PER_SEC;
...
printf ("Avg time for 1 frame: %f \n", (timeSum / (frameNumber - 1)) );
```

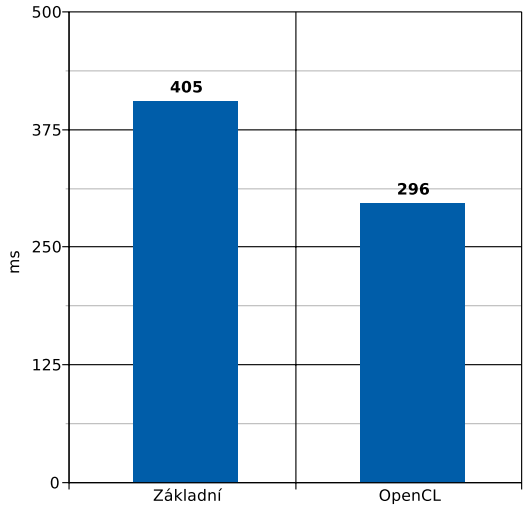
Výpis 4: Měření času vykonávání aktualizace modelu

5.1 Testování 1. videosekvence

Obě implementace algoritmu dosahovaly výkonu zhruba 2-3 snímků za vteřinu. Toto je pro zpracování v reálném čase velmi málo. Obzvlášť na videosekvenci s tak malým rozlišením. Optimalizací implementace by se pravděpodobně dalo dosáhnout výsledku až 5 snímků za vteřinu, což není špatné, ale pro zpracování v reálném čase stále málo.

Implementace musela zpracovat 76 800 obrazových bodů v jednom snímku.

Základní implementace provedla aktualizaci modelu průměrně za 405 ms. Oproti tomu implementace v OpenCL byla díky paralelizaci rychlejší, s časem průměrně 296 ms na snímek, to je zrychlení o 27%. Z toho plyne, že se zařazením výpočetního kernelu do fronty nejsou spojeny žádné další dostatečně velké nároky na výkon, aby znevýhodnili použití OpenCL pro zpracování malých dat na CPU. Výsledky je možné vidět v grafu na obrázku 9.



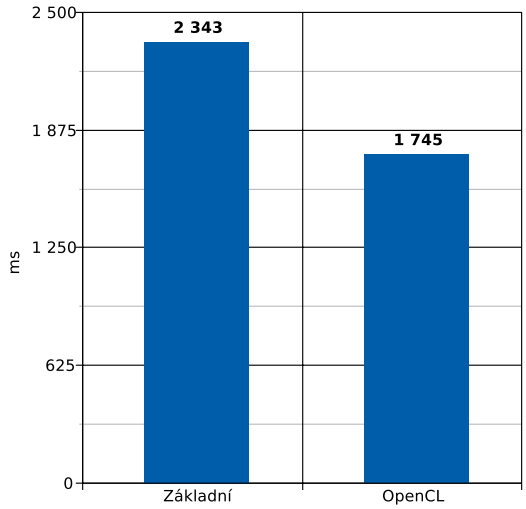
Obrázek 9: Průměrný čas zpracování jednoho snímku videosekvence 1

5.2 Testování 2. videosekvence

Tuto videosekvenci nebyla schopna ani jedna varianta implementace zpracovat pod jednu vteřinu na snímek. Formát druhé videosekvence MPEG (oproti RAW snímkům první videosekvence) neměl na měřený výkon žádný vliv, neboť měření času bylo spuštěno až po načtení snímku. Rychlost je tedy nepřímoúměrná velikosti snímku.

Počet zpracovávaných pixelů v jednom snímku byl 442 368

Implementace bez OpenCL zvládla aktualizovat model průměrně za 2 343 ms na snímek. Algoritmus implementovaný v OpenCL aktualizoval model v průměrném čase 1 745 ms na snímek, což je 26% zrychlení oproti základní variantě.



Obrázek 10: Průměrný čas zpracování jednoho snímku videosekvence 2

6 Závěr

Tato práce představila problematiku rozpoznávání pohybu ve videosekvencích. Rozebrán byl celý průběh od načtení snímku videa až po ověření, zda-li se skutečně jedná o pohybující se objekt ve scéně. Uvedených a krátce popsaných bylo několik metod, které se nejčastěji pro tento úkol používají. Každé nasazení sledování pohybu vyžaduje individuální přístup, protože každá situace má jiné požadavky. Např. notebooky Acer byly pár let zpátky dodáván se softwarem k webkameře, který uměl pracovat v módu „hlídacího psa“. Při jakémkoliv pohybu v záběru spustil nastavený zvuk nebo jinou akci. Pro toto nasazení bude dostačující i metoda rozdílu snímků. Avšak např. pro počítaní projíždějících aut po silnici už bude potřeba lepšího algoritmu.

Takovým může být právě mixtura gausiánů. Tento algoritmus je velice oblíbený a často nasazovaný. Velkou výhodou je multimodálnost, tedy schopnost modelovat pozadí z více objektů v pohledu jednoho pixelu. Takovým příkladem může být pohyb listí na stromě - jeden snímek v pohledu pixelu list je a v dalším už není.

Implementace, která je součástí této práce, je pouze základní metoda. Neobsahuje žádné dodatečné rozšíření, které jsou popsány v kapitole 3.2. Pro reálné nasazení této implementace by bylo vhodné doplnit minimálně proces monitorující náhlé změny osvětlení a morfologický filtr k předcházení šumu v masce popředí, i když tomu se mi podařilo vyhnout nalezením správné kombinace parametrů. Podle druhu použití by případně bylo také vhodné implementovat algoritmus pro vyplňování děr v detekovaných objektech.

Díky implementaci této metody jsem měl možnost se seznámit s velmi zajímavou knihovnou OpenCV, jejíž možností bych v budoucnu ještě rád využil. Zajímavá pro mě byla také myšlenka provádění negrafických výpočtů na grafickém procesoru díky použití OpenCL. Tento standard je poměrně dobře zdokumentován, avšak protože se v době psaní této práce jedná o velmi mladou technologii, není jednoduché najít nějaký dobře popsaný tutoriál, jak s OpenCL začít. V tomto se mi velmi líbila příručka „Průvodce programováním OpenCL pro Mac OS X“ společnosti Apple, která mi pomohla se dostat přeš těžké začátky až ke spuštění prvního kernelu na CPU.

Martin Klapetek

7 Literatura

[1] Stauffer, Grimson. *Adaptive background mixture models for real-time tracking*. Cambridge, 1999.

[2] Power, Schoonees. *Understanding Background Mixture Models for Foreground Segmentation*. Auckland, New Zeland, 2002.

[3] KaewTraKulPong, Bowden. *An Improved Adaptive Background Mixture Model for Real-time Tracking with Shadow Detection*. In Proc. 2nd European Workshop on Advanced Video Based Surveillance Systems, AVBS01. Září 2001.

[4] Karmann, Brandt. *Moving object recognition using and adaptive background memory*. In FRAME-RATE Workshop, IEEE. Elsevier Science Publishers B.V., 1990. s. 751-767.

[5] Toyama, Krumm, Brumitt, Meyers. *Wallflower: Principles and practice of background maintenance*. In ICCV, 1999. s. 255-261.

[6] Cucchiara, Piccardi, Prati. *Detecting moving objects, ghosts, and shadows in video streams*. In IEEE Transactions on Pattern Analysis and Machine Intelligence 25, 2003. s. 1337–1342.

[7] Elgammal, Harwood, Davis. *Non-parametric Model for Background Subtraction*. In Proceedings of IEEE ICCV Frame-rate workshop, 1999.

[8] Heikkila, Silven. *A real-time system for monitoring of cyclists and pedestrians*. In Second IEEE Workshop on Visual Surveillance, (Fort Collins, Colorado), Červen 1999. s. 246–252

[9] Koller, Weber, Malik. *Robust multiple car tracking with occlusion reasoning*. Tech. Rep. UCB/CSD- 93-780, EECS Department, University of California, Berkeley, 1993.

[10] McFarlane, Schofield. *Segmentation and tracking of piglets in images*. Machine Vision and Applications 8(3), 1995. s. 187–193

[11] Fuentes, Velastin. *From tracking to advanced surveillance*. In Proceedings of IEEE International Confer- erence on Image Processing. Barcelona, Španělsko, 2003.